jc582 U.S. PTO
03/23/98

Our Ref.: 83000.1007
P2975

A

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Assistant Commissioner for Patents
BOX PATENT APPLICATION
Washington, D.C. 20231

Sir:

Transmitted herewith for filing is the patent application of:

INVENTOR(S): KAPONO D. CARTER

FOR: METHOD AND APPARATUS FOR SELECTING ATTACHMENTS

Enclosed are:

XX 8 Sheets of Drawings.
XX An Assignment of the Invention to: Sun Microsystems, Inc.
XX Assignment Recordation Cover Sheet - Form PTO 1595.
XX A Declaration and Power of Attorney signed by the inventor Kapono D. Carter.

The Filing Fee has been calculated as shown below:

	(Col. 1)	(Col. 2)	SMALL ENTITY		OTHER THAN A SMALL ENTITY	
For:	No. Filed	No. Extra	Rate	Fee	Rate	Fee
Basic Fee:	[redacted]	[redacted]	[redacted]	$0	[redacted]	$790.00
Total Claims:	23 -20	3	x 11	$0	x 22	$ 66.00
Indep. Claims:	5 -3	2	x 41	$0	x 82	$ 164.00
☐ Multiple Dependent Claim(s) Presented			+135	$0	+270	$
* If the difference in Col. 1 is less than zero, enter "0" in Col. 2			TOTAL	$0	TOTAL	$ 1020.00

____ The Commissioner is hereby authorized to charge payment of any additional fees associated with this communication, or credit any overpayment, to Deposit Account Number 08-1520. A duplicate of this authorization is enclosed.

____ A check in the amount of $______ to cover the Filing Fee is enclosed.

XXX A check in the amount of $40.00 to cover the fee for Recordation of the Assignment is enclosed.

Respectfully submitted,

HECKER & HARRIMAN

[illegible]

J. D. Harriman
Reg. No. 31,967

Dated: March 23, 1998

2029 Century Park East, Suite 1600
Los Angeles, California 90067
(310) 286-0377

CERTIFICATE OF MAILING

This is to certify that this correspondence is being deposited with the United States Postal Service as Express Mail Label No. EM 484490103 US in an envelope addressed to: Assistant Commissioner for Patents, Box Patent Application, Washington, D.C. 20231 on: March 23, 1998.

[illegible]

Signature : Lillian E. Rodriguez

3-23-98
Date

09047684.032398

83000.1007/P2975

UNITED STATES PATENT APPLICATION

FOR

METHOD AND APPARATUS FOR SELECTING ATTACHMENTS

INVENTORS:

KAPONO D. CARTER

PREPARED BY:

HECKER & HARRIMAN
2029 Century Park East
Suite 1600
Los Angeles, CA 90067

(310) 286-0377

CERTIFICATE OF MAILING
This is to certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as Express Mail Label No. EM 484 490 103 US in an envelope addressed to: Assistant Commissioner for Patents, Washington, D.C. 20231 on:

March 23, 1998

[illegible] 3-23-98
Signature Date
Lillian E. Rodriguez

BACKGROUND OF THE INVENTION

1. FIELD OF THE INVENTION

This invention relates to the field of computer software, and, more specifically, to electronic mail applications.

Portions of the disclosure of this patent document contain material that is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure as it appears in the Patent and Trademark Office file or records, but otherwise reserves all copyright rights whatsoever. Sun, Sun Microsystems, the Sun logo, Java, JavaBeans, HotJava and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

2. BACKGROUND ART

With the proliferation of personal computers and communications networks, electronic mail, commonly referred to as "e-mail," has become a popular mechanism for the exchange or distribution of information among individuals, and within or between enterprises, for both private and commercial purposes. An e-mail message may be generated by a "sender" using an e-mail application on the sender's computer system, and transmitted over a communications network to a "receiver" on another computer system. The receiver is able to view the e-mail message using his own e-mail application.

One utility of many e-mail systems is the ability to "attach" previously authored information to an e-mail message. These "attachments" are transmitted with the e-mail message. Common forms of attached information include text files, graphics files and other forms of bounded data. With the development of the World Wide Web (WWW), also referred to simply as "the web", e-mail attachments have expanded to include web pages (e.g., HTML (hypertext markup language) documents) and URL's (universal resource locators) for web pages. These attachments are typically identified during construction of an e-mail message by typing or pasting the name of the file or the URL of the web page in a dialog entry window, or by perusing a file directory in order to select a file from a hierarchical list. Unfortunately, there is no mechanism within the e-mail application for previewing an attachment prior to selection, or for browsing for an attachment. This presents a particular disadvantage when selecting attachments associated with the World Wide Web, as a hierarchical list of linked web pages is typically not available and a user may not know the URL of a specific web page offhand. E-mail applications and the associated drawbacks may be better understood from an overview of electronic mail with reference to a known e-mail application.

Electronic Mail

An e-mail message may be analogized to a posted letter or piece of mail. However, instead of a physical object that is itself physically transported from a sender to a receiver, an e-mail message is an electronic representation that is communicated electronically through a communications network. Examples of communications networks used for communicating e-mail messages include, but are not limited to, tel-com networks, wide area networks (WANs), local area networks (LANs), the Internet, the World Wide Web, intranets, extranets, wireless networks, and other networks over which electronic, digital, and/or analog data may be communicated.

E-mail messages are created, sent, received, and read using a communications program, often referred to as a "mail" or "e-mail" application program. An image of an interface of one e-mail program is illustrated in Figure 2. The example is the interface of the mail module of Netscape Communicator, an internet browser application. The interface consists of a window 200 with a row of control buttons 201-210 across the top, headers 212 - 215 below the buttons, and a field that displays a list of messages. In the example of Figure 2, the list indicates a single message 216 represented by an icon with adjacent text indicating the subject, namely "Meeting on the 20th", from "Sender" and created at 9:20 PM.

The buttons are used to create and modify messages. Button 201, "Get Msg" is used to open a message that has been selected in the list (messages can also be opened by double clicking on them). Button 202, "New Msg" is used to create a new message form that can be completed and mailed to a receiver. "Reply" button 203 is used to generate a reply form to a message that is currently in view. When activated, a message form is generated that has as its address the address of the sender of the message being viewed. Optionally, the reply message may include the entire text of the sender's message. "Forward" button 204 generates a message form that includes the sender's message, but with a blank address, so that the message may be optionally annotated and forwarded to another receiver.

Button 205, "File", is used to save a message into a file in a text or other format. Button 206, "Print", is activated to print the message on an attached printer. "Security" button 208 activates security options for a message such as encryption, use of a digital certificate, or digital signature features, for example. Messages can be deleted by the "Delete" button 209. "Stop" button 210 is used to interrupt or stop operations.

Headers above the message list indicate such information as "Subject" 212, "To/From" 213, "Date" 214, and "Priority" 215. Messages in the list can be sorted by subject, by sender or receiver, by ascending or descending date, by urgency, or by any combination thereof.

An e-mail message generated using the example mail program of Figure 2 is illustrated in Figure 3. The e-mail message 216 includes a palette of buttons 301 - 306, along with buttons 208 and 210 from Figure 2. An address field 307 indicates that the message is being sent to "Receiver@receiver.com". A subject field 308 shows the subject as "Meeting on the 20th". The body of the message is displayed in field 309.

The "Send" button 301 is activated to initiate the transmission of the message from the sender to the receiver. The "Quote" button 302 is used to insert the body of the text from a previous message into the body 309 of a current message. The "Address" button 303 prompts the sender to enter an address of the receiver or to select an address from some stored address book. The "Spelling" button 305 performs a spell check on the message text, and the "Save" button 306 is used to save a message as a text file.

The "Attach" button 304 is used to attach one or more electronic files to the e-mail message. Often a sender wishes to send one or more files to a receiver. One method of sending a file to a receiver would be to copy the information from the file (e.g. the text from a text file) and paste that information into the message field 309 of an e-mail message. If the file is large, this may not be possible. Some e-mail systems have limitations on the size of the body of an e-mail message so that some files may be too large to be entered into the body of an e-mail message. In other cases, the files represent non-text data, such as sound, images, or movies, for example, that cannot be easily pasted into an e-mail message. In such circumstances, the attach feature is used.

When the attach button 304 is activated, a dialogue box appears that allows the sender to navigate through a file system and select files to be attached to the e-mail message. Navigation is typically performed using a hierarchical list of file names. After one or more files are selected, the sender transmits the e-mail message and attached file(s) to a receiver. When the receiver reads the message, there is an indicator that one or more files are attached. The receiver activates the attach button and is presented with a dialogue box that enables the receiver to retrieve the attached file or files and place them somewhere in the receiver's file system.

When a sender wishes to attach HTML documents and other web-based information to an e-mail message, the sender uses the hierarchical list of file names to locate the desired document, or the sender specifies, such as through a text entry mechanism, a URL for each HTML document. Each web page is comprised of one or more separate files in a file system. These files can include, for example, an HTML document and text, graphics and sound files identified by "tags" within the HTML document. Web pages are typically linked to other web pages via embedded URL's. In many cases, dozens of web pages are linked to each other to form a related presentation of data. To send such linked pages using e-mail, each page is attached to an e-mail message for sending to a receiver.

SUMMARY OF THE INVENTION

A method and apparatus for selecting attachments is described. When a sender indicates in an e-mail application or applet that an attachment is to be associated with an e-mail message, an attachment chooser window is presented. The attachment chooser window provides a browser-based graphical user interface (GUI) which allows a sender to browse data resources, such as HTML documents and associated links. An attachment mechanism is provided by which a sender can choose a currently displayed data resource for attachment in an e-mail message. In one embodiment, the attachment mechanism allows a user to select whether the attachment is retrieved and attached to an e-mail message as a resource locator (such as a URL) of the chosen data resource, or whether source data of the data resource is retrieved and attached to the e-mail message as one or more source files.

In an embodiment of the invention, an attachment chooser software component is instantiated by an e-mail application. The attachment chooser software component is configured to provide the graphical user interface for selecting a web-based attachment, and comprises the following components: a menu component configured to control how an attachment is added to an e-mail message, e.g., as a resource locator or as source data; an editable "go-to" text field component configured to identify the resource locator of the currently viewed data resource; a toolbar component providing the basic navigation controls for browsing data resources such as web pages; a browsing component configured to perform parsing and rendering of data resources;

and one or more button components configured to receive input to attach a currently viewed data resource to an e-mail message or to cancel an attachment session.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram of one embodiment of a computer system capable of providing a suitable execution environment for an embodiment of the invention.

Figure 2 is an illustration of a graphic user interface (GUI) for a known e-mail application.

Figure 3 is an illustration of a graphic user interface for e-mail message composition in a known e-mail application.

Figure 4 is an illustration of a graphic user interface for a web attachment chooser in accordance with an embodiment of the invention.

Figure 5A is a block diagram of an attachment mechanism and browsing mechanism in accordance with an embodiment of the invention.

Figure 5B is a block diagram of a software component configuration for a web attachment software component in accordance with an embodiment of the invention.

Figure 6 is a flow diagram of a process for an e-mail application to interact with an attachment chooser component in accordance with an embodiment of the invention.

Figure 7 is a flow diagram of a method for selecting attachments in accordance with an embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

The invention is a method and apparatus for selecting attachments. In the following description, numerous specific details are set forth to provide a more thorough description of embodiments of the invention. It will be apparent, however, to one skilled in the art, that the invention may be practiced without these specific details. In other instances, well known features have not been described in detail so as not to obscure the invention.

An embodiment of the present invention provides for navigation and browsing of data resources, such as text, graphics and audio source data, when selecting attachments for an e-mail message. These data resources may also include source data containing data written in a markup language (e.g., HTML) that may be rendered by a browsing mechanism, and source data containing links such as embedded resource locators (e.g., URL's or other location references to data resources) that may be used by a browsing mechanism to navigate from one data resource to another. Data resources may contain source data in the form of one or more files, documents, streams and other sources of electronic information, or portions thereof.

By providing the ability to view the contents of prospective attachments before they are attached, a sender is able to confirm that the attachment contains the data expected. Also, via the browsing feature, a sender using an embodiment of the invention has access to advanced search engines on the web to further assist in the location of desired attachments. Advanced search features are not available in e-mail systems of the prior art.

In an embodiment of the invention, linked data resources can be traversed through standard browsing operations in order to locate a desired attachment. This is an advantage over navigation of a hierarchical list of file names, because a hierarchical file list does not provide information regarding possible embedded links to other files that may exist for web pages and other types of compound or linked documents. Further, web links via URL typically have little or no correspondence to particular file system hierarchies and often transcend single file systems, necessitating, in systems of the prior art, that the sender have knowledge of the particular URL's or the respective file names and directories for the web page or pages that are to be attached to an e-mail message.

Figure 7 illustrates a method for selecting attachments in accordance with an embodiment of the invention. In step 700, during composition of an e-mail message, when a sender expresses a wish to select an attachment, a graphical user interface (GUI) with browser capability is presented to the sender. If the GUI interface is equipped with an attachment menu, the sender may select the type of the prospective attachment from the menu in step 701. Examples of types of attachment include attachment as a resource locator, such as a URL (universal resource locator), and attachment as source data of the chosen data resource, such as the set of data bits forming a rendered

09045784.032398

HTML document, or web page. An attachment may be retrieved as all source data of a chosen data resource or a subset thereof.

In step 702, the sender uses the browser capability of the GUI interface to navigate through the web using a navigation tool set or by selecting links within displayed data resources such as HTML documents. The browsable web may include the World Wide Web or it may comprise a constrained local web.

If the desired web page or other data resource is not found in step 703, the sender selects "cancel" from the GUI interface in step 706 and dismisses the GUI interface. If a desired web page or other data resource is found in step 703, the sender selects "attach" from the GUI interface and the selected attachment is attached to the e-mail message. If the sender does not wish to select any further attachments in step 705, the user selects "cancel" in step 706 to dismiss the GUI interface. However, if further attachments are desired in step 705, the method returns to step 701.

The following description discloses embodiments of apparatus for implementing the method of selecting attachments disclosed above with respect to Figure 7. Description is provided of a GUI interface in accordance with an embodiment of the invention. The discussion of the GUI interface embodiment is followed by description of software apparatus configured to implement the GUI interface and provide for selection of attachments in accordance with an embodiment of the invention. First, however, description is given below of an embodiment of computer apparatus suitable for providing an execution environment for the software apparatus of the invention.

Embodiment of Computer Execution Environment (Hardware)

An embodiment of the invention can be implemented as computer software in the form of computer readable code executed on a general purpose computer such as computer 100 illustrated in Figure 1, or in the form of bytecode class files executable within a Java™ runtime environment running on such a computer. A keyboard 110 and mouse 111 are coupled to a bi-directional system bus 118. The keyboard and mouse are for introducing user input to the computer system and communicating that user input to processor 113. Other suitable input devices may be used in addition to, or in place of, the mouse 111 and keyboard 110. I/O (input/output) unit 119 coupled to bi-directional system bus 118 represents such I/O elements as a printer, A/V (audio/video) I/O, etc.

Computer 100 includes a video memory 114, main memory 115 and mass storage 112, all coupled to bi-directional system bus 118 along with keyboard 110, mouse 111 and processor 113. The mass storage 112 may include both fixed and removable media, such as magnetic, optical or magnetic optical storage systems or any other available mass storage technology. Bus 118 may contain, for example, thirty-two address lines for addressing video memory 114 or main memory 115. The system bus 118 also includes, for example, a 32-bit data bus for transferring data between and among the components, such as processor 113, main memory 115, video memory 114 and mass storage 112. Alternatively, multiplex data/address lines may be used instead of separate data and address lines.

In one embodiment of the invention, the processor 113 is a microprocessor manufactured by Motorola, such as the 680X0 processor or a microprocessor manufactured by Intel, such as the 80X86, or Pentium processor, or a SPARC™ microprocessor from Sun Microsystems™, Inc. However, any other suitable microprocessor or microcomputer may be utilized. Main memory 115 is comprised of dynamic random access memory (DRAM). Video memory 114 is a dual-ported video random access memory. One port of the video memory 114 is coupled to video amplifier 116. The video amplifier 116 is used to drive the cathode ray tube (CRT) raster monitor 117. Video amplifier 116 is well known in the art and may be implemented by any suitable apparatus. This circuitry converts pixel data stored in video memory 114 to a raster signal suitable for use by monitor 117. Monitor 117 is a type of monitor suitable for displaying graphic images.

Computer 100 may also include a communication interface 120 coupled to bus 118. Communication interface 120 provides a two-way data communication coupling via a network link 121 to a local network 122. For example, if communication interface 120 is an integrated services digital network (ISDN) card or a modem, communication interface 120 provides a data communication connection to the corresponding type of telephone line, which comprises part of network link 121. If communication interface 120 is a local area network (LAN) card, communication interface 120 provides a data communication connection via network link 121 to a compatible LAN. Wireless links are also possible. In any such implementation, communication interface 120 sends and receives electrical, electromagnetic or optical signals which carry digital data streams representing various types of information.

Network link 121 typically provides data communication through one or more networks to other data devices. For example, network link 121 may provide a connection through local network 122 to local server computer 123 or to data equipment operated by an Internet Service Provider (ISP) 124. ISP 124 in turn provides data communication services through the world wide packet data communication network now commonly referred to as the "Internet" 125. Local network 122 and Internet 125 both use electrical, electromagnetic or optical signals which carry digital data streams. The signals through the various networks and the signals on network link 121 and through communication interface 120, which carry the digital data to and from computer 100, are exemplary forms of carrier waves transporting the information.

Computer 100 can send messages and receive data, including program code, through the network(s), network link 121, and communication interface 120. In the Internet example, remote server computer 126 might transmit a requested code for an application program through Internet 125, ISP 124, local network 122 and communication interface 120. In accord with the invention, one such downloaded application is the apparatus for selecting attachments described herein.

The received code may be executed by processor 113 as it is received, and/or stored in mass storage 112, or other non-volatile storage for later execution. In this manner, computer 100 may obtain application code in the form of a carrier wave.

Application code may be embodied in any form of computer program product. A computer program product comprises a medium configured to store or transport computer readable code, or in which computer readable code may be embedded. Some examples of computer program products are CD-ROM disks, ROM cards, floppy disks, magnetic tapes, computer hard drives, servers on a network, and carrier waves.

The computer systems described above are for purposes of example only. An embodiment of the invention may be implemented in any type of computer system or programming or processing environment.

Embodiment of an Attachment Chooser Graphical User Interface (GUI)

In an embodiment of the invention, a sender initiates an attachment session by pressing a button or selecting an item from a menu (e.g., an "attach" button or menu item) presented by an e-mail application during composition of an e-mail message. The pressing of the button or selecting of the menu item in the e-mail application generates an event which initiates an attachment session and triggers the presentation of a new window or frame providing an attachment chooser GUI interface. Other mechanisms configured to invoke the attachment chooser GUI interface may be used as well.

An embodiment of an attachment chooser GUI interface in accordance with an embodiment of the invention is illustrated in Figure 4. The attachment chooser GUI interface ("chooser interface") is configured as a web browser. To facilitate browsing, the chooser interface contains display region 400 in which the rendered data, such as HTML data, for a current web page is displayed. Display region 400 is configured as a scrollable display with vertical scrollbar 401A and horizontal scrollbar 401B. A "GoTo" field 402 is provided on the chooser interface which contains the resource locator (e.g., URL) of the current web page. "GoTo" field 402 is editable, permitting the sender to specify a particular resource locator. Entering a resource locator into "GoTo" field 402 causes the chooser interface to display the associated data resource (e.g., an associated web page is displayed in display region 400).

The chooser interface further comprises navigation toolbar 411, which is configured with basic navigation controls in the form of "previous page" button 403, "next page" button 404, "home" button 405, "reload" button 406 and "stop" button 407. Menu 409 provides a mechanism by which the sender can specify whether attachments selected with the chooser interface are to be attached as a simple resource locator or as the source data of the data resource associated with the resource locator. Menu 409 is configured with selectable states, with the currently selected state indicating the form an attachment will take (i.e., the "attachment type'). The current state of menu 409 in Figure 4 specifies that attachments are to be attached as URL's.

Several control buttons are also provided on the chooser interface. "Attach" button 408, when activated, is configured to return an attachment associated with the currently displayed data resource to the e-mail application for attachment to the e-mail message being composed. The state of menu 409 is queried, when "attach" button 408 is activated, to determine how the attachment is to be returned to the e-mail application, e.g., as a URL or as an input stream containing the bits of the source data associated with the URL. "Cancel" (or "Done") button 410 is configured to dismiss the chooser interface when activated, ending the attachment session. A "help" button (not shown) may also be provided which, when activated, causes a web page containing chooser interface-related help information to be displayed in display region 400.

When the chooser interface is first displayed, the sender's home page (as specified in user preferences or by a default URL) is displayed in display region 400. Because there is no resource locator history when the attachment session begins, the previous/next page navigation buttons 403 and 404 are disabled. At this point, the sender can either type a new resource locator into "GoTo" field 402, or select a resource locator link contained in the displayed home page. When a link is selected, the chooser interface automatically displays the page or data resource associated with the selected link. The resource locator is extracted, and "GoTo" field 402 is updated.

When "previous page" button 403 is activated (e.g., when a mouse button is depressed while a mouse cursor is above button 403), a resource locator history is accessed to change the currently displayed data resource to the previously displayed data resource. Similarly, when "next page" button 404 is activated, the resource locator history is accessed to change the currently displayed data resource to the next data resource. "Previous page" button 403 is enabled once a resource locator history is established by browsing at least two data resources. When "previous page" button 403 is used to navigate backward through a resource locator history, "next page" button 404 is enabled to permit navigation forward through the resource locator history.

"Home" button 405 is configured to cause the sender's home page to be displayed in display region 400. "Reload" button 406 is configured to reload the current web page (or other current data resource). Reloading is particularly useful when the current web page contains data that is dynamically updated. "Stop" button 407 is configured to halt the loading of the current web page. "Stop" button 407 can be used to halt the download of large, undesired graphics files, or to stop animated graphics.

Embodiment of Software Apparatus for Selecting Attachments

An embodiment of the invention includes software apparatus comprising a component or collection of components configured to support an attachment chooser GUI interface, such as the chooser interface previously described with reference to Figure 4. The components may be implemented as instances of object classes in accordance with known object-oriented programming practices, or the components may be implemented under one or more component model definitions. Several component model definitions are currently available, such as COM, CORBA, and the Java™ component scheme referred to as JavaBeans™.

Each component model provides for encapsulation of related functions and data structures into individual components, similar to what occurs under a standard object-oriented programming (OOP) approach. The particular mechanisms by which the components are managed and interact are defined according to the respective component model. Bridges (e.g., ActiveX) may be constructed which allow components designed under different component model definitions to interact within a single application. Interaction is typically performed through a set of methods implemented by the component. These sets of methods are referred to as "interfaces" in some component models. The public methods by which OOP object classes interact are often presented in the form of application programming interface (API) definitions.

To provide a better understanding of encapsulation of related data structures and methods, an overview of object-oriented programming is provided below.

Object-Oriented Programming

Object-oriented programming is a method of creating computer programs by combining certain fundamental building blocks, and creating relationships among and between the building blocks. The building blocks in object-oriented programming systems are called "objects." An object is a programming unit that groups together a data structure (one or more instance variables) and the operations (methods) that can use or affect that data. Thus, an object consists of data and one or more operations or procedures that can be performed on that data. The joining of data and operations into a unitary building block is called "encapsulation."

An object can be instructed to perform one of its methods when it receives a "message." A message is a command or instruction sent to the object to execute a certain method. A message consists of a method selection (e.g., method name) and a plurality of arguments. A message tells the receiving object what operations to perform.

One advantage of object-oriented programming is the way in which methods are invoked. When a message is sent to an object, it is not necessary for the message to instruct the object how to perform a certain method. It is only necessary to request that the object execute the method. This greatly simplifies program development.

Object-oriented programming languages are predominantly based on a "class" scheme. The class-based object-oriented programming scheme is generally described in Lieberman, "Using Prototypical Objects to Implement Shared Behavior in Object-Oriented Systems," OOPSLA 86 Proceedings, September 1986, pp. 214-223.

A class defines a type of object that typically includes both variables and methods for the class. An object class is used to create a particular instance of an object. An instance of an object class includes the variables and methods defined for the class. Multiple instances of the same class can be created from an object class. Each instance that is created from the object class is said to be of the same type or class.

To illustrate, an employee object class can include "name" and "salary" instance variables and a "set_salary" method. Instances of the employee object class can be created, or instantiated for each employee in an organization. Each object instance is said to be of type "employee." Each employee object instance includes "name" and "salary" instance variables and the "set_salary" method. The values associated with the "name" and "salary" variables in each employee object instance contain the name and salary of an employee in the organization. A message can be sent to an employee's employee object instance to invoke the "set_salary" method to modify the employee's salary (i.e., the value associated with the "salary" variable in the employee's employee object).

A hierarchy of classes can be defined such that an object class definition has one or more subclasses. A subclass inherits its parent's (and grandparent's etc.) definition. Each subclass in the hierarchy may add to or modify the behavior specified by its parent class. Some object-oriented programming languages support multiple inheritance where a subclass may inherit a class definition from more than one parent class. Other programming languages support only single inheritance, where a subclass is limited to inheriting the class definition of only one parent class. The Java™ programming language also provides a mechanism known as an "interface" which comprises a set of constant and abstract method declarations. An object class can implement the abstract methods defined in an interface. Both single and multiple inheritance are available to an interface. That is, an interface can inherit an interface definition from more than one parent interface.

An object is a generic term that is used in the object-oriented programming environment to refer to a module that contains related code and variables. A software application can be written using an object-oriented programming language whereby the program's functionality is implemented using objects. As previously discussed, the encapsulation provided by objects in an object-oriented programming environment may be extended to the notion of components under a component model definition.

Implementation in the Java^{TM} Programming Language

An embodiment of the software apparatus of the invention is implemented in the Java^{TM} programming language. The Java^{TM} programming language is an object-oriented programming language with each program comprising one or more object classes. Unlike many programming languages, in which a program is compiled into machine-dependent, executable program code, Java^{TM} classes are compiled into machine independent bytecode class files. Each class contains code and data in a platform-independent format called the class file format. The computer system acting as the execution vehicle supports the Java^{TM} runtime environment. The runtime environment contains a program called a virtual machine, which is responsible for executing the code in Java^{TM} classes.

Applications may be designed as standalone Java^{TM} applications, or as Java^{TM} "applets" which are identified by an applet tag in an HTML document, and loaded by a browser application. The class files associated with an application or applet may be stored on the local computing system, or on a server accessible over a network. Each class is loaded into the Java^{TM} runtime environment, as needed, by the "class loader."

Java^{TM} classes are loaded on demand from the network (stored on a server), or from a local file system, when first referenced during an application or applet's execution. The runtime environment locates and loads each class file, parses the class file format, allocates memory for the class's various components, and links the class with other already loaded classes. This process makes the code in the class readily executable by the virtual machine.

Java™ classes may also be incorporated into Java™ components referred to as "JavaBeans™". JavaBeans™ are designed in accordance with the JavaBean™ API Specification to allow for component-based application building. Bridges (e.g., ActiveX bridges) may be used with JavaBeans™ to allow JavaBeans™ to be used in other component model environments, such as OLE/COM and CORBA.

Support for features such as "introspection," "customization," "events," "properties" and "persistence" is provided within the JavaBean™ framework to facilitate application building and component use. "Introspection" permits builder tools to analyze how a particular bean works. "Customization" permits an application builder to customize the appearance and behavior of a bean. "Events" provide a simple communication metaphor that can be used to connect a bean with other application components or beans. "Properties" are used for bean customization and programmatic use. "Persistence" allows for a bean to have its customized state saved and reloaded later. These features are discussed in the JavaBean™ API Specification, Version 1.01, by Sun Microsystems (1997), which is available on the World Wide Web at the URL, "http://java.sun.com/beans/spec.html", and is incorporated herein by reference.

Embodiments of the software apparatus may be implemented using standard OOP object classes or using components under a known component model definition. For the purposes of the following description, references to components may refer to instances of OOP object classes or components under a known component model.

Implementation of Software Apparatus

Figure 5A is a block diagram of an apparatus for selecting attachments in accordance with an embodiment of the invention. The apparatus comprises an attachment mechanism 501 and a browsing mechanism 502 coupled to an e-mail application 500, wherein each mechanism may comprise hardware, software, firmware and/or any other suitably-configured element. Browsing mechanism 502 is configured to display data resources to a sender for consideration as attachments in an e-mail message under composition within e-mail application 500. Browsing mechanism is also configured to respond to user input to navigate between data resources, such as through resource locators embedded in the displayed data resources.

Attachment mechanism 501 is configured to respond to user input selecting a currently displayed data resource for attachment by retrieving the attachment from browsing mechanism 502, and providing the attachment to e-mail application 500. The attachment may be retrieved in the form of different attachment types. For example, the attachment may be retrieved as a resource locator associated with the currently displayed data resource, or as all of or a subset of the source data of the currently displayed data resource. Attachment mechanism 501 may be further configured to respond to user input to select the attachment type.

An embodiment of the apparatus of Figure 5A is illustrated in Figure 5B in the form of software components. In the embodiment of Figure 5B, attachment mechanism is implemented as attachment chooser 501, whereas browsing mechanism 502 is implemented as HTML browsing component 502. The embodiment of Figure 5B is directed to HTML-based data resources, though it will be obvious to one skilled in the art that embodiments of the invention may be similarly implemented for other forms of data resources.

Figure 5B illustrates software apparatus for implementing an attachment chooser GUI interface in accordance with an embodiment of the invention. The software apparatus include e-mail application (or applet) 500 and attachment chooser component 501. E-mail application 500 and attachment chooser component 501 interact via messages and events. Attachment chooser component 501 comprises components 502-510, which are configured to implement the individual features of the attachment chooser GUI interface that are related to browsing. Attachment chooser component 501 further comprises components 511-514, which assist in the interaction between e-mail application 500 and attachment chooser component 501, or provide control over attachment chooser component 501.

The individual components of attachment chooser component 501 comprise multiple button components 505-506, 508-510 and 513-514 corresponding to the displayed buttons on the chooser GUI interface illustrated in Figure 4. The button components receive a "buttonPushed" event when the corresponding graphic representation of the button on the chooser GUI interface is clicked (i.e., a mouse button is pressed while a cursor is within the bounds of the button representation). Receipt of the buttonPushed event can be used to trigger a particular action or set of actions. For the case of browser-related button components 505-506 and 508-510, the buttonPushed event is used to invoke a corresponding method of HTML browsing component 502, wherein methods for navigating the web and for parsing and rendering HTML data are implemented.

Component 511 is an "attach" menu in support of element 409 of Figure 4. "Attach" menu 511 allows a user to select a current menu state from a set of possible menu states. For this embodiment, the set of possible menu states include "URL" and "InputStream" (or "Source"), which define the form an attachment is to take (i.e., the "attachment type") when the attachment is selected. "Attach" menu 511 includes an access method by which attachment chooser component 501 can query the menu state at the time an attachment is selected.

Component 507 is an editable "goto" textfield in support of element 402 of Figure 4. Textfield 507 displays the URL of the current web page. In order to update the displayed URL, textfield 507 receives events from HTML browsing component 502 when the web page displayed by HTML browsing component 502 changes, for example, through selection of a link within a displayed web page. Also, textfield 507 can receive user input specifying a new URL, in which case, textfield 507 invokes an access method of HTML browsing component 502, such as setDocumentURL(URL url), to update the current document property in the HTML browsing component 502. Invoking the access method of HTML browsing component 502 causes the browsing component to navigate to the new URL.

Component 512 is a callback list that includes each component or application that registers with attachment chooser component 501 as an "ActionListener." Callback list 512 is used by "attach" button 514 to deliver events when the "attach" button 514 (supporting element 408 of Figure 4) is activated by a buttonPushed event. For this embodiment, e-mail application 500 is part of callback list 512, and is therefore informed through a callback event when the "attach" button is activated. E-mail application 500 responds to the callback event by retrieving the attachment from attachment chooser component 501 through an API call.

Component 513 is a "cancel" button in support of element 410 of Figure 4. When a buttonPushed event is received, "cancel" button 513 dismisses attachment chooser component 501, removing the chooser GUI interface from display. In some embodiments, "cancel" button 513 may also be used to initiate a shutdown of attachment chooser component 501, including the release of memory allocated to attachment chooser component 501 and its associated underlying components.

As previously stated, methods for navigating and for parsing and rendering HTML data are implemented within HTML browsing component 502. In Figure 5B, an HTML parsing and rendering component 503 and a document stack component 504 cooperate to form HTML browsing component 502. HTML parsing and rendering component 503 derives from a scrollable panel class and is configured to support the display region 400 and scrollbars 401A and 401B of Figure 4. Document stack 504 maintains the browser history in the form of a stack of URL's, providing the record necessary for the forward and back navigation familiar to web browsers. When a link is selected within a web page, HTML parsing and rendering component 503 navigates to the new web page designated by the link, and sends an event to document stack 504 , so that document stack 504 knows to push the URL of the new web page onto the stack.

Component 505 is a "previous" button in support of element 403 of Figure 4. "Previous" button 505 is set to activate a previousDocument() method of document stack 504 when a buttonPushed event is received. In response, document stack 504 sends an event to HTML parsing and rendering component 503 indicating that it should go to the previous web page. Component 506 is a "next" button in support of element 404 of Figure 4. "Next" button 506 is set to activate a nextDocument() method of document stack 504 when a buttonPushed event is received. In response, document stack 504 sends an event to HTML parsing and rendering component 503 indicating that it should go to the next web page. Properties within document stack 504 indicate whether a "previous page" or a "next page" are available. Access methods for these properties may be used to control the enablement of "previous" button 505 and "next" button 506.

Components 508-510 are a "home" button, "reload" button and "stop" button in support of elements 405, 406 and 407, respectively, of Figure 4. Similarly to buttons 505 and 506, buttons 508-510 are set to activate corresponding methods within HTML browsing component 502 when a buttonPushed event is received. Specifically, button 508 invokes the setDocumentURL(URL url) method, designating the sender's home page URL; button 509 invokes a reload() method; and button 510 invokes a stop() method. In this embodiment, the setDocumentURL(), reload() and stop() methods are implemented in HTML parsing and rendering component 503.

In an embodiment of the invention, the button, menu and textfield components may be implemented using pre-defined object classes from the Java™ Abstract Windowing Toolkit (AWT) package, including a textfield class, a menu class, and a button class. Also, in an embodiment of the invention, the HTML parsing and rendering component 503 and document stack 504 may be implemented with the HotJava™ HTML Component and the Document Stack Bean specified for the HotJava™ Browser. Documentation for the Document Stack Bean and the HotJava HTML Component are provided herein as Appendix A. The respective API's of the

Document Stack Bean and the HotJava HTML Component may be referenced therein. It will be obvious to one skilled in the art that other known or user-defined components or combinations of components may also be used to implement the components of the software apparatus illustrated in Figure 5B.

Attachment chooser component 501 presents the following API to e-mail application 500:

```
public void addActionListener(ActionListener I)
        - Registers a callback when the "attach" button component is
        activated.

public void removeActionListener(ActionListener I)
        - Deregister the "attach" button component callback.

public Object getAttachment()
        - Returns either a URL or an InputStream, depending on the
        state of the menu component.

public String getName()
        - Returns the name of the attachment.

public void setURL(URL u)
        - Explicitly sets the location displayed by the chooser GUI
        interface.
```

The addActionListener() and removeActionListener() are methods available to e-mail application 500 (and other components) to subscribe or unsubscribe to callback list 512. Those components that are registered as an ActionListener will receive a callback event when "attach" button 514 is activated. Receipt of the callback event generated by "attach" button 514 indicates that the sender has selected the current web page as an attachment.

The getAttachment() method is used by e-mail application 500 to retrieve an attachment after receipt of a callback event from "attach" button 514. When getAttachment() is called, attachment chooser component 501 queries the menu state of "attach" menu 511 to determine the attachment type. If the menu state indicates "URL," then attachment chooser component 501 invokes the getDocumentURL() method of HTML browsing component 502 to extract and return the URL of the current web page. If the menu state indicates "InputStream" (or "Source"), then attachment chooser component 501 invokes the getDocumentSource() method of HTML browsing component 502 to extract and return the source data of the current web page as an input stream. This scheme can be expanded to similarly include attachments in the form of selections from a web page (e.g., user highlighted portions of a page), or in the form of indicated elements whose value is a link under a cursor.

The getName() method is used by e-mail application 500 to extract the name or title of a current document to associate with an attachment previously extracted using the getAttachment() method. The setURL() method provides a mechanism by which the e-mail application can specify a URL to be viewed.

Figure 6 is a flow diagram of a process whereby an e-mail application instantiates an attachment chooser component and retrieves an attachment in accordance with an embodiment of the invention. In step 600, the e-mail application obtains an instance of the attachment chooser component. Step

600 may comprise calling a constructor method of the attachment chooser component. In step 601, the e-mail application registers itself as a listener on the attachment chooser component, such as by invoking the addActionListener() method of the attachment chooser component. In step 602, the e-mail application obtains an instance of a frame, and, in step 603, adds the attachment chooser component to the frame to display the GUI interface of the attachment chooser component.

At step 604, the e-mail application waits for receipt of a callback event from the attachment chooser component. After the callback event is received, indicating that the sender has selected an attachment, the e-mail application retrieves the attachment from the attachment chooser component in step 605, for example, by invoking a getAttachment() method. In step 606, a branching occurs based on whether the retrieved attachment object is an object of type URL or an object of type InputStream. If the attachment is a URL object, in step 607, the attachment is included as a URL in the e-mail message under composition. If the attachment is an InputStream object, in step 608, the attachment is included as source files in the e-mail message.

Thus, a method and apparatus for selecting attachments has been described in conjunction with one or more specific embodiments. The invention is defined by the claims and their full scope of equivalents.

APPENDIX A

The HotJava™ Component Series

HotJava Browser Components

Overview

About 80 percent of the functionality of the HotJava Browser is provided by four JavaBeans components: one very large and the other three quite simple. These components may be used anywhere that the display or browsing of documentation is useful: from e-mail systems and help viewers to data reporting and kiosks.

By far the richest component of the four is the HotJava HTML Component. It provides the ability to parse HTML and then render it in a panel subject to a large number of configuration properties. Setting its idea of the "current document" by one of several methods will cause navigation to the specified URL. The HotJava Document Stack Bean primarily allows the "forward" and "back" navigation familiar to browsers. The stack listens for events from components like pushbuttons in order to manipulate its idea of "current position" among the visited documents. The HotJava Authenticator Bean may simply be added to the same container to allow connections to secure sites. The HotJava System State Bean provides the interface by which users may access system properties of the HTML component.

This document is a jumping-off point for developers assumed to be familiar with JavaBeans. The following reference pages are intended for both "component assemblers" who will be instantiating and connecting the beans in builder applications as well as Java developers who will be programatically accessing the components. If terms like property, event, method, introspection, or serialization are unfamiliar, you may want to refer to the JavaBeans documentation and take the JavaBeans tutorial.

Example Use

For an example of using the components see Getting Started with the HTML Component in the BeanBox.

Note

On Windows there may exist a problem where the JDK's JRE (Java Runtime Environment) does not consistently derive the correct home directory. The JDK uses HOME and the JRE composes the directory from HOMEDRIVE and HOMEPATH. To solve this, it is recommended that you explicitly set these environment variables to be consistent.
For example:

HOME=c:\home
HOMEDRIVE=c:
HOMEPATH=\home

Component Reference Pages

Authenticator Bean	- allows connections to secure sites
Document Stack Bean	- forward and back navigation among visited documents
HotJava HTML Component	- HTML parsing and rendering in a Panel
HotJava System State Bean	- allows connections to secure sites

See Also

HotJava Delegating Security Manager	- allows a developer-provided security manager to co-exist with the HotJava HTML Component's security manager

---

HotJava Components version 1.1.2

Copyright © 1996 Sun Microsystems, Inc. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure as it appears in the Patent and Trademark Office file or records, but otherwise reserves all copyright rights whatsoever.

The HotJava™ Component Series

HotJava Document Stack Bean

Class At A Glance

Class Name: HotjavaDocumentStack
Extends: Object
Implements: Serializable, BrowserHistoryListener

Purpose

The HotJavaDocumentStack is an invisible JavaBean that maintains a stack of URLs describing "documents." It is intended for use with the HotJava HTML Component. It is typically combined with user interface controls to provide the "forward" and "back" behavior seen in HotJava and other Web browsers.

Key Properties

logicalDepth	- How many documents deep is the stack?
contentsDepth	- How many documents deep is document state kept?
nextDocumentAvailable	- is there a next document for which to enable a "Forward" button?
previousDocumentAvailable	- is there a previous document for which to enable a "Back" button?

Creation

HotjavaDocumentStack()

Commonly Used Methods

executeHistoryCommand() - process a history command.
nextDocument() - go forward to the next document.
previousDocument() - go back to the previous document.

About This Document

Audience

This document describes the HotJava Document Stack JavaBean component. It is intended for both "component assemblers" who will be instantiating and connecting the bean in builder applications as well as Java developers who will be programatically accessing the component. It assumes that you are already familiar with JavaBeans. If terms like property, event, method, introspection, or serialization are unfamiliar, you may want to refer to the JavaBeans documentation or take the JavaBeans tutorial.

Table of Contents

Class Description

Overview

The document stack is an invisible bean intended for use with the HotJava HTML Component. It provides the record necessary for the forward" and "back" navigation familiar to Web browsers.

The document stack bean is simply a collection of the "documents" visited by the user. It can process a number of history commands to push a document on the stack or clear the history. Methods are provided to go to the previous and next documents, and to indicate the existence of next and previous to allow enabling and disabling controls as appropriate.

Introspection

A BeanInfo class, HotjavaDocumentStackBeanInfo, explicitly provides information for properties, events, and methods.

Example Use

Instantiate a HotjavaDocumentStack, HotJava HTML Component, and two buttons. Label one button "Forward" and bind its buttonPushed event to the nextDocument() method of the document stack. Label the other "Back" and set it to activate previousDocument(). Connect the executeHistoryCommand event for HTML component to the corresponding method of the document stack. Do the same in reverse from the executeHistoryCommand event of the document stack to the method of the HTML component.

Summaries

Property Summary

boolean	logicalDepth	- How many documents deep is the stack?	(rw)
boolean	contentsDepth	- How many documents deep is document state kept?	(rw)
boolean	nextDocumentAvailable	[bound] - is there a next document?	(rw)
boolean	previousDocumentAvailable	[bound] - is there a previous document?	rw)

Method Summary

instantiation	HotjavaDocumentStack()

navigation	nextDocument()
	previousDocument()

state accessors	isNextAvailable()
	setNextAvailable()
	isPreviousAvailable()
	setPreviousAvailable()

adding/removing document history	eraseDocumentHistory()
	executeHistoryCommand()

adding/removing listeners	addBrowserHistoryListener()
	removeBrowserHistoryListener()
	addPropertyChangeListener()
	removePropertyChangeListener()

Event Summary

As Source

EventListener Interfaces: PropertyChangeListener
BrowserHistoryListener

As Listener

EventListener Interfaces: BrowserHistoryListener

Alphabetical Reference

Properties

contentsDepth
    Type: int
    Get: getContentsDepth()
    Set: setContentsDepth()

logicalDepth
    Type: int
    Get: getLogicalDepth()
    Set: setLogicalDepth()

nextDocumentAvailable [bound]
    Type: boolean
    Get: isNextAvailable()
    Set: setNextAvailable()

previousDocumentAvailable [bound]
    Type: boolean
    Get: isPreviousAvailable()
    Set: setPreviousAvailable()

Methods

void addBrowserHistoryListener(BrowserHistoryListener l)
    Add this listener to be sent executeHistoryCommand() messages.
    See also: removeBrowserHistoryListener().

void addPropertyChangeListener(PropertyChangeListener l)
    Add this listener to be notified of changes to bound properties.
    See also: removePropertyChangeListener().

void eraseDocumentHistory()
    Fires a BrowserHistoryEvent with id equal to Clear to indicate that browser should erase the document history.

void executeHistoryCommand(BrowserHistoryEvent evt)
    Processes a BrowserHistoryEvent. This method is implemented as part of the BrowserHistoryListener interface. It is used to enable the stack bean to receive history notifications from the browser bean. Make no assumptions about the internals of this event, i.e. treat this an an opaque event.

int getContentsDepth()

Gets the value of the contentsDepth property.
See also: setContentsDepth().

int getLogicalDepth()
Gets the value of the logicalDepth property.
See also: setLogicalDepth().

HotjavaDocumentStack()
Default constructor.

boolean isNextAvailable()
Get the nextDocumentAvailable property.
See also: setNextAvailable().

boolean isPreviousAvailable()
Get the previousDocumentAvailable property.
See also: setPreviousAvailable().

void nextDocument()
Fires a BrowserHistoryEvent with id equal to Next to indicate that browser should go to the next document.

void previousDocument()
Fires a BrowserHistoryEvent with id equal to Previous to indicate that browser should go to the previous document.

void removeBrowserHistoryListener(BrowserHistoryListener l)
Remove this listener so it no longer receives executeHistoryCommand() messages.
See also: addBrowserHistoryListener().

void removePropertyChangeListener(PropertyChangeListener l)
Remove this listener from those notified of changes on bound properties.
See also: addPropertyChangeListener().

void setContentsDepth(boolean val)
Set the contentsDepth property. This property controls how far back on the stack the browser will hold the state associated with a web page.
Examples of document state include running applets, and the contents of HTML forms. Setting this to a large number will increase memory consumption. The default value is 10.
See also: getContentsDepth().

void setLogicalDepth(boolean val)
Set the logicalDepth property. This property controls how large the stack is allowed to grow. The default value is 100.
See also: getLogicalDepth().

void setNextAvailable(boolean val)
Set the nextDocumentAvailable property.
See also: isNextAvailable().

void setPreviousAvailable(boolean val)
Set the previousDocumentAvailable property.
See also: isPreviousAvailable().

Events

As Source

The two document stack properties are bound. Therefore, notification of change to next/previous document availablity is multicast with a PropertyChangedEvent to any PropertyChangeListener registered using addPropertyChangeListener().

BrowserHistoryEvent objects are sent to registered BrowserHistoryListeners to notify of changes in the "current document" due to nextDocument() and previousDocument() actions.

EventListener Interfaces:	PropertyChangeListener
	BrowserHistoryListener
EventObjects:	PropertyChangeEvent
	BrowserHistoryEvent

As Listener

The document stack receives executeHistoryCommand() messages in order to add documents as they are visited in the browser and to clear its history when requested.

EventListener Interfaces:	BrowserHistoryListener
EventObjects:	BrowserHistoryEvent

See Also

Related Topics

HotJava Browser Components
HotJava Authenticator Bean
HotJava HTML Component
HotJava System State Bean

HotJava Components version 1.1.2

Copyright © 1996 Sun Microsystems, Inc. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure as it appears in the Patent and Trademark Office file or records, but otherwise reserves all copyright rights whatsoever.

The HotJava™ Component Series

HotJava HTML Component

Class At A Glance

Class Name: HotJavaBrowserBean
Extends: BeanDocumentPanel
Implements: Externalizable, ComponentListener, BrowserHistoryListener,
Observer, Serializable

Purpose

The HotJava HTML Component is the encapsulation of about 80 percent of what a web browser does. Specifically, this is the HTML parsing and rendering engine used in the HotJava Browser. This component is a candidate for use in any application that requires display of documents or navigation similar to that found in web browsers.

Key Properties

documentString - the URL for the current document as a string.

Creation

HotJavaBrowserBean()
clone()

Commonly Used Methods

setDocumentString() - set (and display) a document by specifying aURL as a string.
find() - search for a given string in the current document.
print()- print the current document.

About This Document

Audience

This document describes the HTML JavaBean component. It is intended for both "component assemblers" who will be instantiating and connecting the bean in builder applications as well as Java developers who will be programatically accessing the component. It assumes that you are already familiar with JavaBeans. If terms like property, event, method, introspection, or serialization are unfamiliar, you may want to refer to the JavaBeans documentation or take the JavaBeans tutorial.

Table of Contents

Class Description

Overview

The HotJava HTML Component is a single, monolithic panel class that complies with the JavaBeans 1.0 specification. While the major browser functionality is contained within this single bean, there are also the Authenticator, Document Stack, and System State beans which provide supporting functionality.

There is also a normal Java class behind the bean that implements the methods. This class could also be used in Java applications. The danger with this is upward compatibility. If in the future browser functionality is provided as a collection of beans, rather than one, then the panel class becomes obsolete. Compatibility with the monolithic bean component will be maintained via bean aggregation, but the panel class, if used in Java applications, will have no such guarantee.

To support the possible future development directions, the intent is to keep this API as minimal as possible, while providing a full featured product. This is a difficult balance between making it easier to use, and providing an API that won't need to be broken in the future.

Security

There are two types of security concerns with the HTML component:
The security manager, which determines what it can do, and dealing with secure connections.

An HTML component presents a difficult situation. The Bean specification says that security management is the responsibility of the bean container. Beans that are to be used everywhere need to be able to run under a security manager, while beans that are application specific may not need to. The HTML component is unique in that it is a bean that can have other beans or Java applets within it that can come from unknown sources, i.e. the web. If the HTML component were to just let its container worry about security, as recommended by the Bean specification, you could easily get into a situation where you are browsing the web and running applets totally unprotected. To protect against this, when the bean is instantiated, if it doesn't detect a security manager it will install one. The installed security manager will only affect applets within the bean. This still allows applets to take advantage of a lenient container-supplied security manager, but it does provide good default security behavior. We also provide an option for the case where an application needs to have its own security manager in addition to the HTML Component's security manager. The application can have both managers running simultaneously by instantiating the DelegatingSecurityManager. The only drawback to this method is that the DelegatingSecurityManager assumes that the HTML Component Jar file is on the CLASSPATH.

To deal with secure connections, we need to handle authentication. Dealing with authentication means supplying a user interface, something we are trying to avoid in the bean. As a compromise, we are breaking the monolithic model, and supplying the HotJava Authenticator Bean. This release defines the interface used by this bean, and provides a default implementation. Developers may use this version or override its behavior. If the authenticator bean is not wired into the container, the HTML component will work fine, but it will not be able to connect to secure sites.

09045854.032598

Providing a Pop-Up Menu

The right mouse pop-up menu is not implemented in the HTML component itself. If the client wishes to provide this functionality, they can listen for a right mouse button down event, display their own menu, and use the menu select information with the IndicatedElement property to perform any requested function.

Document History and Navigation

Also note that this bean doesn't directly support a history list or forward/back behavior. That functionality may be provided by the client, or by using the HotJava Document Stack.

Introspection

A BeanInfo class, HotJavaBrowserBeanBeanInfo, explicitly provides information for properties, events, and methods.

Customization

Rather than providing a BeanCustomizer class, the relevant state of the HTML component may be accessed through the use of the HotJava System State bean.

Summaries

Property Summary

CurrentDocument	currentDocument - the document being displayed	[constrained][bound]	(rw)
String	documentString - the URL for the current document as a string	[bound]	(rw)
URL	documentURL - the URL for the current document	[bound]	(rw)
Reader	documentSource - the input stream for the current document	[bound]	(rw)
String	documentTitle - the title of the current document	[bound]	(ro)
String	errorMessage - the most recent error	[bound]	(ro)
String	statusMessage - the latest status	[bound]	(ro)
double	loadingProgress - progress information during document loading	[bound]	(ro)
boolean	documentReloadable - true when the document can be reloaded	[bound]	(ro)
boolean	secureConnection - true when the current connection is secure	[bound]	(ro)
String[]	frameList - the names of all HTML frames	[bound] indexed	(ro)
DocumentSelection	selection - the user's current selection	[bound]	(ro)
ElementInfo	indicatedElement - the link (if any) under the cursor	[bound]	(ro)
String	charset - character set used to translate into Unicode	[bound]	(rw)

Method Summary

instantiation	HotJavaBrowserBean()
	clone()
control	find()
	print() (2)
	clearImageCache()
	executeHistoryCommand()
	reload()
	stopLoading()
	start()
	stop()
accessing the document	getCurrentDocument()
	setCurrentDocument()
	getDocumentString()
	setDocumentString()
	getDocumentURL()
	setDocumentURL()
	getDocumentSource()
	setDocumentSource()
reading status	getDocumentTitle()
	getErrorMessage()
	getStatusMessage()
	getLoadingProgress()
	getSelection()
	getIndicatedElement()
	isDocumentReloadable()
	isSecureConnection()
accessing frames	getFrameList() (2)
accessing character set	getCharset()
	setCharset()
adding/removing listeners	addBrowserHistoryListener()
	removeBrowserHistoryListener()
	addPropertyChangeListener()
	removePropertyChangeListener()
	addVetoableChangeListener()
	removeVetoableChangeListener()

Event Summary

As Source

EventListener Interfaces:	BrowserHistoryListener
	PropertyChangeListener
	VetoableChangeListener

As Listener

EventListener Interfaces:	BrowserHistoryListener
	ComponentListener

Alphabetical Reference

Properties

charset [bound] - character set used to translate into Unicode
Type: String
Get: getCharset()
Set: setCharset()

currentDocument [constrained][bound] - the document being displayed
Type: CurrentDocument
Get: getCurrentDocument()
Set: setCurrentDocument()

documentReloadable [bound] - true when the document can be reloaded
Type: boolean
Get: isDocumentReloadable()
Set: none

documentSource [bound] - the input stream for the current document
Type: Reader
Get: getDocumentSource()
Set: setDocumentSource()

documentString [bound] - the URL for the current document as a string
Type: String
Get: getDocumentString()
Set: setDocumentString()

documentTitle [bound] - the title of the current document
Type: String
Get: getDocumentTitle()

Set: none

documentURL [bound] - the URL for the current document
Type: URL
Get: getDocumentURL()
Set: setDocumentURL()

errorMessage [bound] - the most recent error string
Type: String
Get: getErrorMessage()
Set: none

frameList [bound] indexed - the names of all HTML frames
Type: String[]
Get: getFrameList()
Set: none

indicatedElement [bound] - the link (if any) under the cursor
Type: ElementInfo
Get: getIndicatedElement()
Set: none

loadingProgress [bound] - the most recent progress information during document loading
Type: double
Get: getLoadingProgress()
Set: none

secureConnection [bound] - true when the current connection is secure
Type: boolean
Get: isSecureConnection()
Set: none

selection [bound] - the user's current selection
Type: DocumentSelection
Get: getSelection()
Set: none

statusMessage [bound] - the most recent status message
Type: String
Get: getStatusMessage()
Set: none

Methods

void addBrowserHistoryListener(BrowserHistoryListener l)
Add a listener to receive executeHistoryCommand messages.
See also: removeBrowserHistoryListener(), addPropertyChangeListener(), removePropertyChangeListener(), addVetoableChangeListener(), removeVetoableChangeListener().

void addPropertyChangeListener(PropertyChangeListener l)
Add a listener for post-notification of changes in a bound property.
See also: removePropertyChangeListener(), addBrowserHistoryListener(), removeBrowserHistoryListener(), addVetoableChangeListener(), removeVetoableChangeListener().

void addVetoableChangeListener(VetoableChangeListener l)
Add a listener for pre-notification of changes in a constrained property.
See also: removeVetoableChangeListener(), addBrowserHistoryListener(), removeBrowserHistoryListener(), addPropertyChangeListener(), removePropertyChangeListener().

void clearImageCache()
Clear the image cache. Calling this method will dump all cached images.

HotJavaBrowserBean clone()
This method clones the current browser panel and returns it to the client. This is intended to be used in response to user requests for a new browser window. If the client wishes to support this feature, they can do so by cloning the browser panel, and adding it to a new Frame. This way all context, POST information, etc. is preserved.
See also: HotJavaBrowserBean().

void executeHistoryCommand(BrowserHistoryEvent evt)
Processes a BrowserHistoryEvent. This method is implemented as part of the BrowserHistoryListener interface. It is used to enable the stack bean to receive history notifications from the browser bean. Make no assumptions about the internals of this event i.e treat this an an opaque event.

int find(int pos, String val)
Searches the current document for text matching val starting at position pos. Position 0 is the beginning of the document. The document is then scrolled to bring the first match into view, and the position of that match is returned, and can be passed to subsequent calls to find the next occurrence. -1 is returned if no match is found. Other then the special values mentioned, the client should not make any assumptions about the meaning of position.

String getCharset()
Get the charset property. This is a bound property, normally set by a MIME header in the HTTP connection. This setting affects the character set that is used to translate the incoming byte stream into Unicode.
See also: setCharset().

CurrentDocument getCurrentDocument()
Get the currentDocument property: a class representing the currently viewed document. The currentDocument property may be null if the browser is not currently viewing a specific location.
See also: setCurrentDocument(), getDocumentString(), getDocumentURL().

Reader getDocumentSource()

Get the HTML source for the current document.
See also: setDocumentSource().

String getDocumentString()
Get the documentString property: the URL of the currently viewed document as a string. It may be null if the browser is not currently viewing a specific location.
See also: setDocumentString(), getDocumentURL(), getCurrentDocument().

String getDocumentTitle()
Get the documentTitle property. This is a bound property, set when loading a new document to the value specified by the TITLE HTML tag. May be null.

URL getDocumentURL()
Get the documentURL property: the URL of the currently viewed document. It may be null if the browser is not currently viewing a specific location.
See also: setDocumentURL(), getDocumentString(), getCurrentDocument().

String getErrorMessage()
Get the errorMessage property. This is a bound property, set whenever an error is encountered.

String[] getFrameList()
Get the frameList property. This is a bound, indexed property. When frames are created or destroyed, their names are added or removed from this list. This information is mostly useful for clients who are maintaining multiple bean instances, each with its own set of frames, as a document in one may try to force navigation in a frame in another.

String getFrameList(int index)
Get a specific frame by index from the frameList.

ElementInfo getIndicatedElement()
Get the indicatedElement property. This is a bound property whose value is the link (if any) under the cursor. The ElementInfo class contains the inline image URL (if any), the href URL (if any), the alternate text (if any), and the MouseEvent data that triggered the change (Up, Down, or Moved if the links change).
See also: getSelection().

double getLoadingProgress()
Get the loadingProgress property. This is a bound property used to provide progress information during the loading of a document, and its components. The value returned is a double precision floating point number that will vary between 0 and 1. This value will indicate a composite of the load state of all the different items being loaded. This property will always be set to 0 at the start of loading, and to 1 when loading finishes.
See also: isDocumentReloadable() reload().

DocumentSelection getSelection()
Get the selection property. This is a bound property, set whenever the user highlights a portion of the displayed document. The return value is a DocumentSelection containing the selected area in text and raw HTML.

See also: getIndicatedElement().

String getStatusMessage()
Get the statusMessage property. This is a bound property, set whenever the current status changes.

HotJavaBrowserBean()
Default constructor. Used to construct all instances of the browser bean. If a security manager is to be installed, it will be done here. The browser panel is ultimately derived from ScrollPanel, so it is scrollable.
See also: clone().

boolean isDocumentReloadable()
Is the current document reloadable? Returns true if there is a current, valid document. This is a bound property.
See also: reload().

boolean isSecureConnection()
Get the secureConnection property. This is a bound property, set whenever the security of the connection changes. It is true when the current connection is secure.

void print()
Prints the contents of the browser panel. Intended to be invoked directly by controls like a "Print" button.

void print(PrintJob job)
Prints the contents of the browser panel, with additional parameters determined by the PrintJob.

void reload()
Reload the current document if possible. Ignored if there is no document to reload.
See also: isDocumentReloadable(), getLoadingProgress(), stopLoading().

void removeBrowserHistoryListener(BrowserHistoryListener l)
Remove a listener that had been receiving executeHistoryCommand messages. addPropertyChangeListener(), removePropertyChangeListener(), addVetoableChangeListener(), removeVetoableChangeListener().
See also: addBrowserHistoryListener().

void removePropertyChangeListener(PropertyChangeListener l)
Remove a listener for post-notification of changes in a bound property.
See also: addPropertyChangeListener(), addBrowserHistoryListener(), removeBrowserHistoryListener(), addVetoableChangeListener(), removeVetoableChangeListener().

void removeVetoableChangeListener(VetoableChangeListener l)
Remove a listener for pre-notification of changes in a constrained property.
See also: addVetoableChangeListener(), addBrowserHistoryListener(), removeBrowserHistoryListener(), addPropertyChangeListener(), removePropertyChangeListener().

void setCharset(String charset)
Set the charset property. This is a bound property, normally set by a MIME header in the HTTP connection. This setting affects the character set that is used to translate the incoming byte stream into Unicode.
See also: getCharset().

void setCurrentDocument(CurrentDocument doc)
Set the currentDocument property: a class containing the String, URL, frame name of the currently viewed document, and a hint whether or not a new window should be created with a cloned bean. This last value is a hint only, and will always be false when received as a bound property change. It will have no effect in the bean itself. Setting this property has the side effect of initiating navigation to the new URL. This is a constrained property, so the change of state can be vetoed by the listener. If it is not vetoed the document at that location will be fetched and displayed in the frame named, if it exists. If the frame name is not null, and does not exist in the bean, then all frames will be removed, and the document displayed, with the frame given the indicated name. This change should be listened to by clients who might wish to create another bean instance for the indicated frame, rather than using this one, based on the supplied hint, or their own criteria. If the client has already created other bean instances, with named frames, and the frame named in this property matches one, then this property should be vetoed, and set in the other bean.
See also: getCurrentDocument(), getDocumentString(), getDocumentURL().

void setDocumentSource(Reader source)
Set the document source property. This is a bound property that can be used to cause HTML source to be rendered. When this method is used directly (as opposed to being set as part of the navigation process), the document and documentURL properties are first set to null. If these changes are not vetoed, then HTML source from the Reader will be parsed and rendered.
See also: getDocumentSource().

void setDocumentString(String url)
Set the documentString property: the URL of the currently viewed document as a string. This has the side effect of initiating navigation to the new URL. This is merely a bound property, but setting it will trigger setting of the documentURL property, and the currentDocument property, which is vetoable.
See also: getDocumentString(), setDocumentURL(), setCurrentDocument().

void setDocumentURL(URL url)
Set the documentURL property: the URL of the currently viewed document. This has a side effect of initiating navigation to the new URL. This is a bound property, but setting it will trigger setting of the documentString (if it indicates a different URL) property, and the currentDocument property, which is vetoable.
See also: getDocumentURL(), getDocumentString(), getCurrentDocument().

void stopLoading()
Stop loading of current document if possible. Ignored if done loading.
See also: getLoadingProgress(), reload().

void stop()

Stop animations, or stop formatting the current page. Useful if the container holding the bean is hidden from view.
See also: start().

void start()
Start animations on the current page and/or finish formatting the current page.
See also: stop().

Events

As Source

Most events are either pre-notifications of property change with the option for the listener to veto (constrained), or post-notifcations of property change (bound). Listeners may register for pre-notification of constrained properties with addVetoableChangeListener() and/or post-notifcation of bound properties with addPropertyChangeListener().

Communication with the HotJava Document Stack is handled by means of the BrowserHistoryListener interface. Messages of the form executeHistoryCommand() are sent to the document stack to notify of changes in the current document state or to clear the history.

EventListener Interfaces:	BrowserHistoryListener
	PropertyChangeListener
	VetoableChangeListener
EventObjects:	BrowserHistoryEvent
	PropertyChangeEvent

As Listener

The HotJava Document Stack also sends the executeHistoryCommand() method to update the HTML component as "next" and "previous" operations are performed. Hence, this component is also a BrowserHistoryListener.

This component is also intended to be the target of button pushed and other component activation, and therefore implements ComponentListener.

EventListener Interfaces:	BrowserHistoryListener
	ComponentListener
EventObjects:	BrowserHistoryEvent

Supporting Classes

```
class CurrentDocument {
public String documentString;  // String rep of URL
 public URL documentURL;        // URL
public String frameName;       // Frame document is displayed in
public boolean externalHint;   // A hint given if the HTML component
                               // thinks the client should veto the change, and
                               // set this property on a new bean instance,
                               // or an existing instance that contains
                               // the named frame
};

class Cookie {
public int version;            // Cookie version
public String attribute;       // Attribute name
public String value;           // Value
public String domain;          // Domain
public String path;            // Path
public String comment;
public Date expires;           // Exipration date
public boolean secure;         // Is this a secure cookie?
public static Cookie parseFromString( String cookieString );
public static Cookie[] parseFromString( String cookieString );
public static String[] combineAsStrings( Cookie[] cookieList );
public String toString();      // Convert to string.
};

class CookieJar {
public CookieJar(Cookie[]);
public Cookie[] getCookies();
public boolean isIncrementalChange(CookieJar other);
public Cookie entryAdded(CookieJar other);
public Cookie entryRemoved(CookieJar other);
};

class PoolEntry {
public String url;             // URL string of entry
public Date lastTouched;       // Expiration date of entry
public String URL;             // URL string of entry
};

class URLPool {
public URLPool(PoolEntry[]);
public PoolEntry[] Entries();
public boolean isIncrementalChange(URLPool other);
public PoolEntry entryAdded(URLPool other);
public PoolEntry entryRemoved(URLPool other);
};

class DocumentSelection {
public String text;            // Text of selection
public String html;            // HTML of selection
};

class ElementInfo {
```

```
public String imageURL;       // Image link
public String hrefURL;        // HREF link
public String altText;        // Alt text
public MouseEvent event;      // source event
};

class SystemProperties {
public SystemProperties(Hashtable values)
public Hashtable getValues()
}
```

See Also

Related Topics

HotJava Browser Components
HotJava Authenticator Bean
HotJava Document Stack Bean
HotJava System State Bean
DelegatingSecurityManager

HotJava Components version 1.1.2

Copyright © 1996 Sun Microsystems, Inc. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure as it appears in the Patent and Trademark Office file or records, but otherwise reserves all copyright rights whatsoever.

CLAIMS

1. An apparatus comprising:

a browsing mechanism configured to render a current data resource in a display region of a graphical user interface, said browsing mechanism configured to navigate through a plurality of data resources; and

an attachment mechanism configured to retrieve an attachment from said browsing mechanism in response to a user event, said attachment associated with said current data resource.

2. The apparatus of claim 1, wherein said attachment comprises a resource locator associated with said current data resource.

3. The apparatus of claim 1, wherein said attachment comprises source data associated with said current data resource.

4. The apparatus of claim 1 wherein said attachment mechanism is configured to select an attachment type of said attachment.

5. The apparatus of claim 1, wherein said attachment mechanism comprises a button on said graphical user interface.

6. The apparatus of claim 1, wherein said browsing mechanism is configured to navigate to a first data resource using a resource locator in a second data resource.

7. A method for selecting attachments comprising:

displaying a graphical user interface having a browsing mechanism configured to render a data resource;

browsing through one or more data resources using said browsing mechanism to determine a desired data resource; and

retrieving an attachment from said browsing mechanism, said attachment associated with said desired data resource.

8. The method of claim 7 further comprising the step of selecting a type of said attachment.

9. The method of claim 7 wherein said step of retrieving said attachment comprises retrieving a resource locator of said desired data resource.

10. The method of claim 7 wherein said step of retrieving said attachment comprises retrieving source data associated with said desired data resource.

11. The method of claim 7, wherein said step of browsing comprises the step of navigating a resource locator in said one or more documents.

12. A computer program product comprising:

a computer usable medium having computer readable code embodied therein for selecting an attachment, said computer program product comprising:

computer readable code configured to cause a computer to display a graphical user interface having a browsing mechanism configured to render a data resource;

computer readable code configured to cause a computer to respond to user input to browse through one or more data resources using said browsing mechanism; and

computer readable code configured to cause a computer to retrieve an attachment from said browsing mechanism, said attachment associated with a desired data resource.

13. The computer program product of claim 12 further comprising computer readable code configured to cause a computer to receive user input to select a type of said attachment.

14. The computer program product of claim 12 wherein said computer readable code configured to cause a computer to retrieve said attachment comprises computer readable code configured to cause a computer to retrieve a resource locator of said desired data resource.

15. The computer program product of claim 12 wherein said computer readable code configured to cause a computer to retrieve said attachment comprises computer readable code configured to cause a computer to retrieve source data associated with said desired data resource.

16. The computer program product of claim 12, wherein said computer readable code configured to cause a computer to respond to user input to browse comprises computer readable code configured to cause a computer to navigate a resource locator in said one or more documents.

17. A memory configured to store data for access by a computer system, comprising:

a data structure stored in said memory and associated with a graphical user interface, said data structure comprising:

a browsing component comprising:

one or more methods configured to render a current data resource;

one or more navigation methods configured to navigate between a plurality of data resources;

one or more navigation components configured to invoke said one or more navigation methods of said browsing component in response to user input; and

an attachment component comprising a method configured to retrieve an attachment from said browsing component in response to a user input, said attachment associated with a desired data resource.

18. The memory of claim 17, wherein said attachment comprises a resource locator of said desired data resource.

19. The memory of claim 17, wherein said attachment comprises source data associated with said data resource.

20. The memory of claim 17, wherein said data structure further comprises:

a property which determines a type of said attachment; and

a selection method configured to allow a user to select a value of said property.

21. The memory of claim 17, wherein said one or more navigation methods are configured to navigate a resource locator in a data resource in response to a user input.

22. The memory of claim 17, wherein said browsing component further comprises:

a stack configured to contain resource locators of navigated data resources;

one or more methods configured to browse said navigated data resources by stepping forward and backward within said stack.

23. An apparatus comprising:

a browsing means for rendering a current data resource in a display region of a graphical user interface, said browsing means for navigating through a plurality of data resources; and

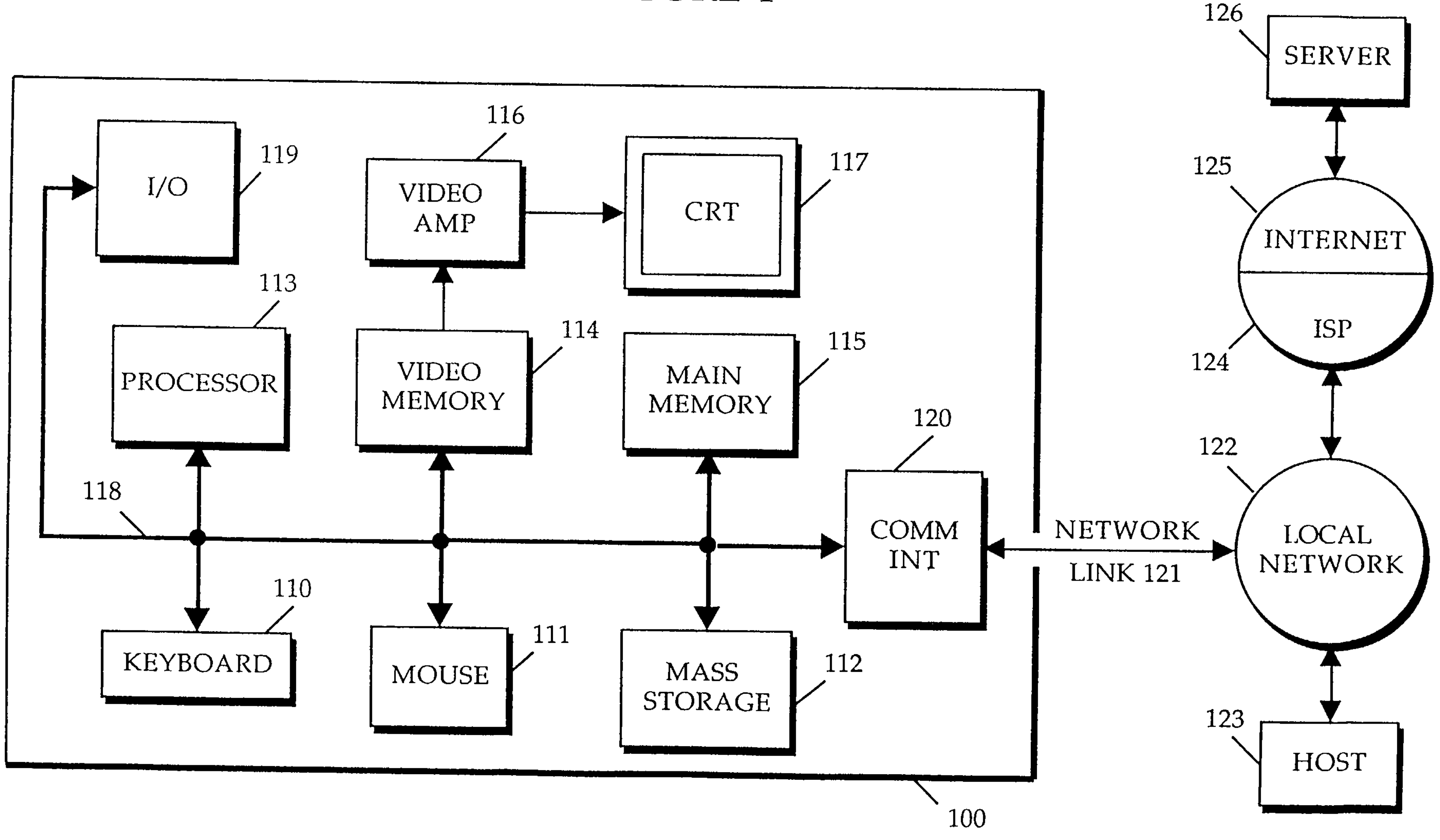
means for retrieving an attachment from said browsing means in response to a user event, said attachment associated with said current data resource.

09047834.032598

ABSTRACT OF THE DISCLOSURE

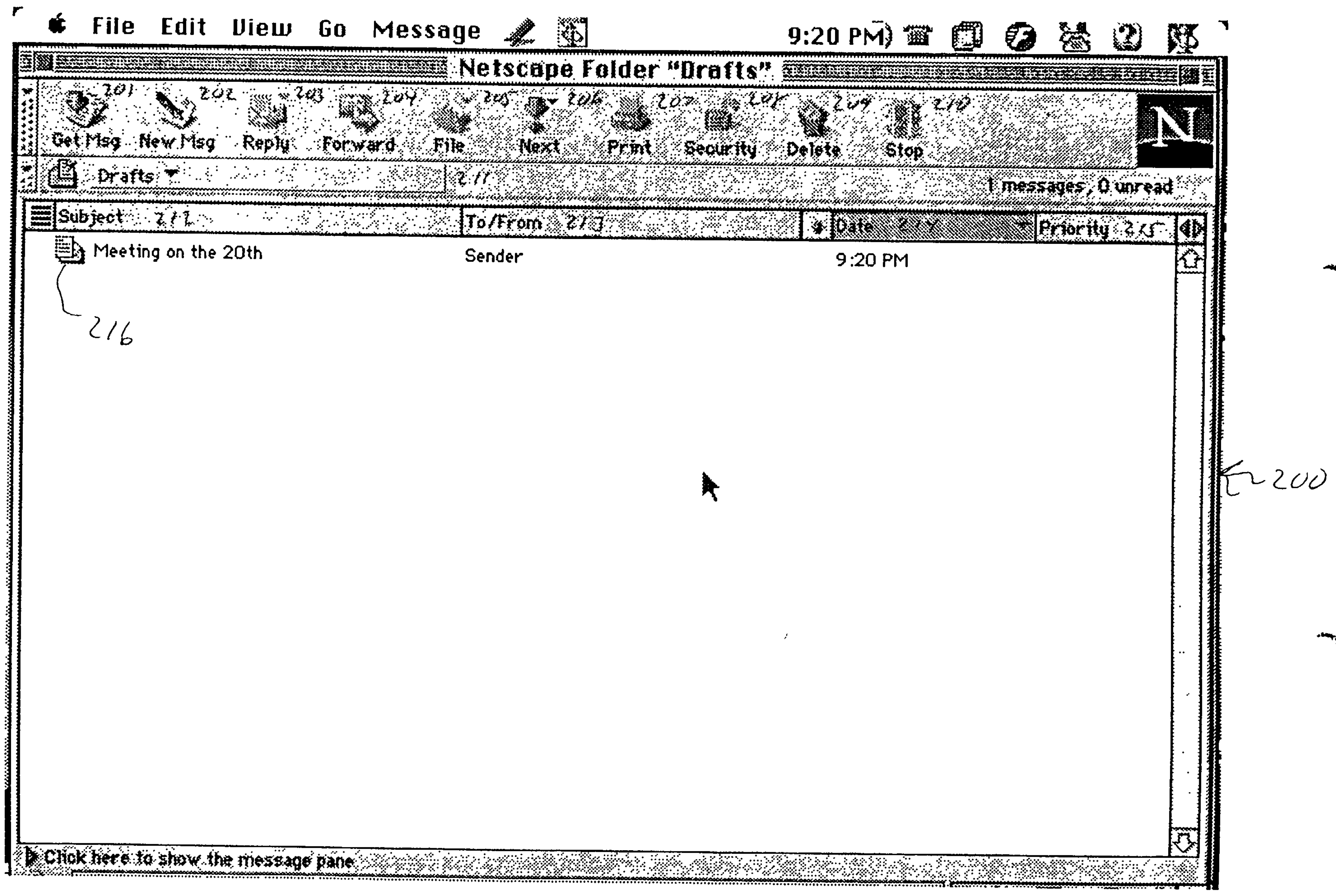
A method and apparatus for selecting attachments. When a sender indicates in an e-mail application or applet that an attachment is to be associated with an e-mail message, an attachment chooser window is presented. The attachment chooser window provides a browser-based graphical user interface (GUI) which allows a sender to browse data resources, such as HTML documents and associated links. An attachment mechanism is provided by which a sender can choose a currently displayed data resource for attachment in an e-mail message. In one embodiment, the attachment mechanism allows a user to select whether the attachment is retrieved and attached to an e-mail message as a resource locator (such as a URL) of the chosen data resource, or whether source data of the data resource is retrieved and attached to the e-mail message as one or more source files.

FIGURE 1



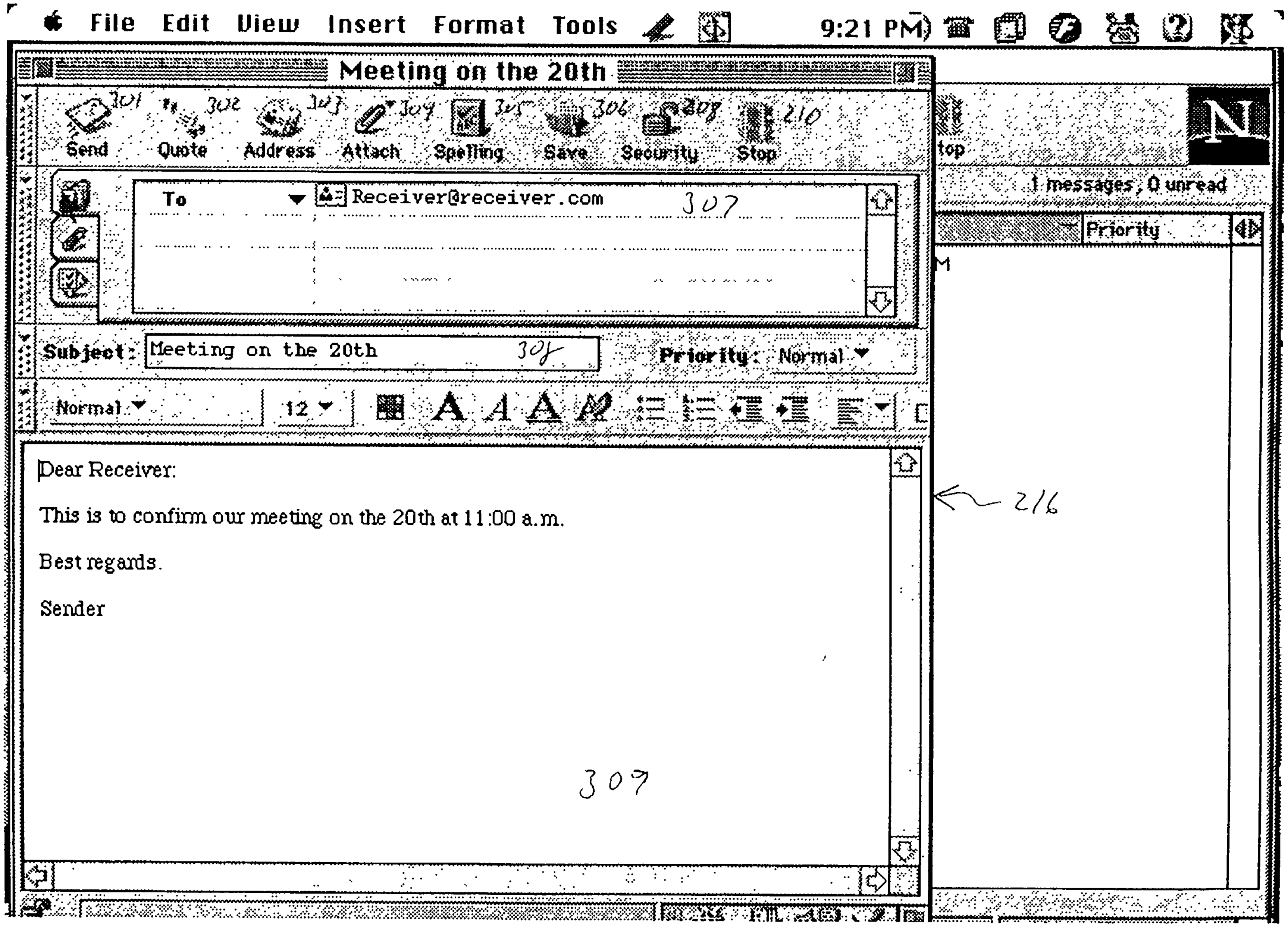
I/O
119
116
VIDEO AMP
CRT
117
113
PROCESSOR
VIDEO MEMORY
114
MAIN MEMORY
115
120
COMM INT
118
110
KEYBOARD
MOUSE
111
MASS STORAGE
112
100
NETWORK LINK 121
126
SERVER
125
INTERNET
ISP
124
122
LOCAL NETWORK
123
HOST

09046784.032398



Subject	To/From	Date	Priority
Meeting on the 20th	Sender	9:20 PM	

Figure 2



To: Receiver@receiver.com

Subject: Meeting on the 20th

Priority: Normal

Normal 12

Dear Receiver:

This is to confirm our meeting on the 20th at 11:00 a.m.

Best regards.

Sender

Figure 3

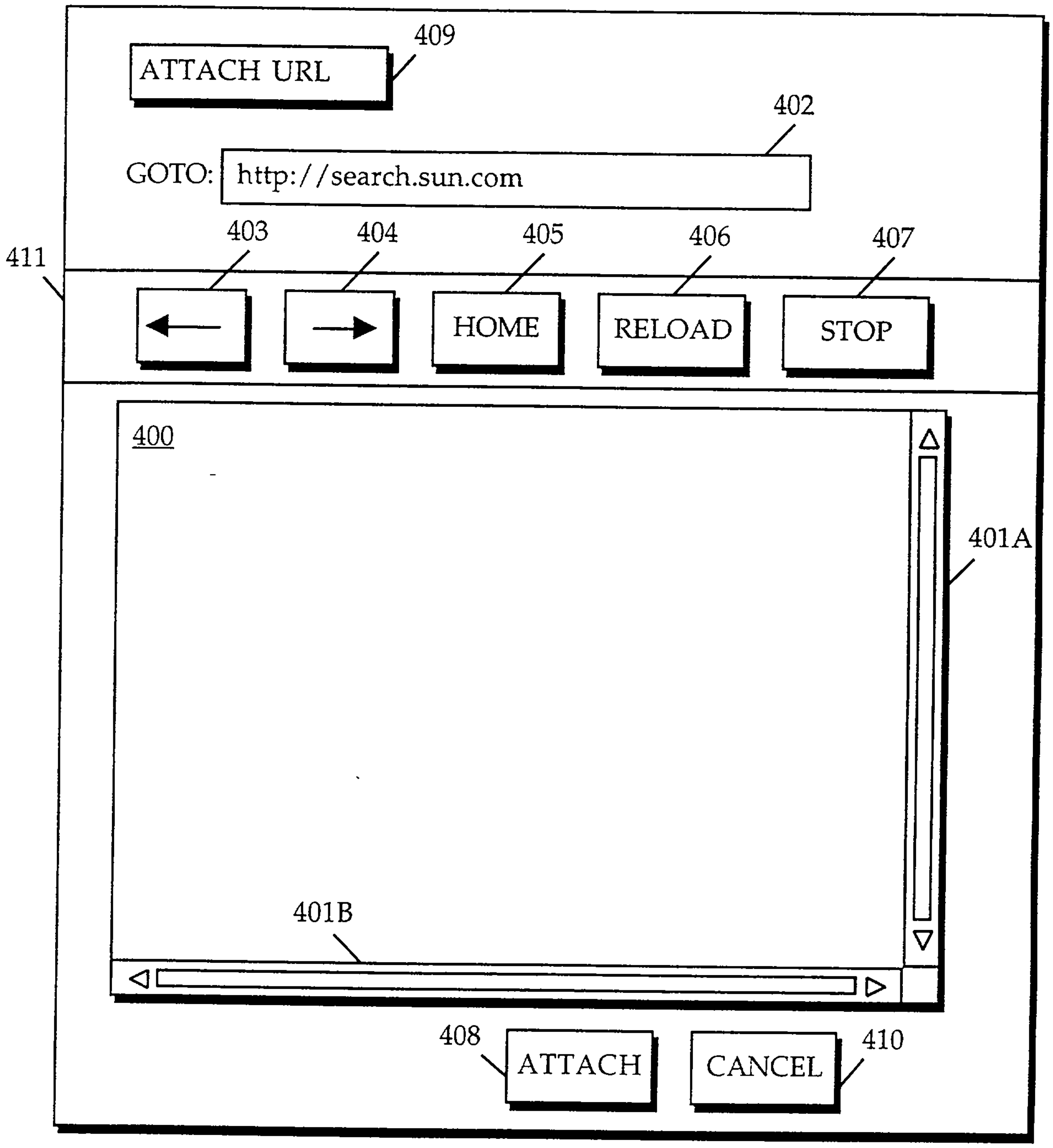


FIGURE 4

09523034.030600

09625724.062800

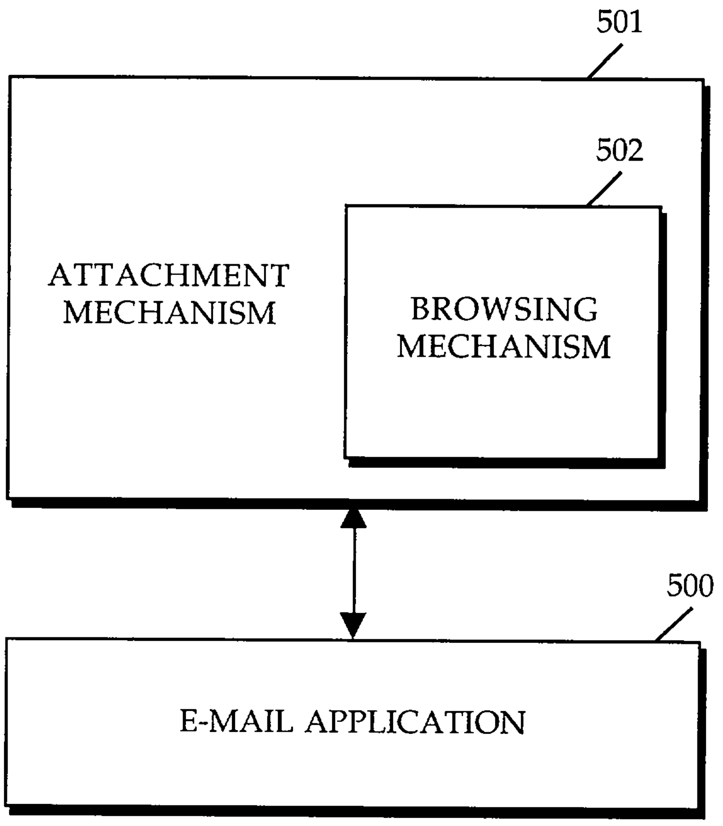


FIGURE 5A

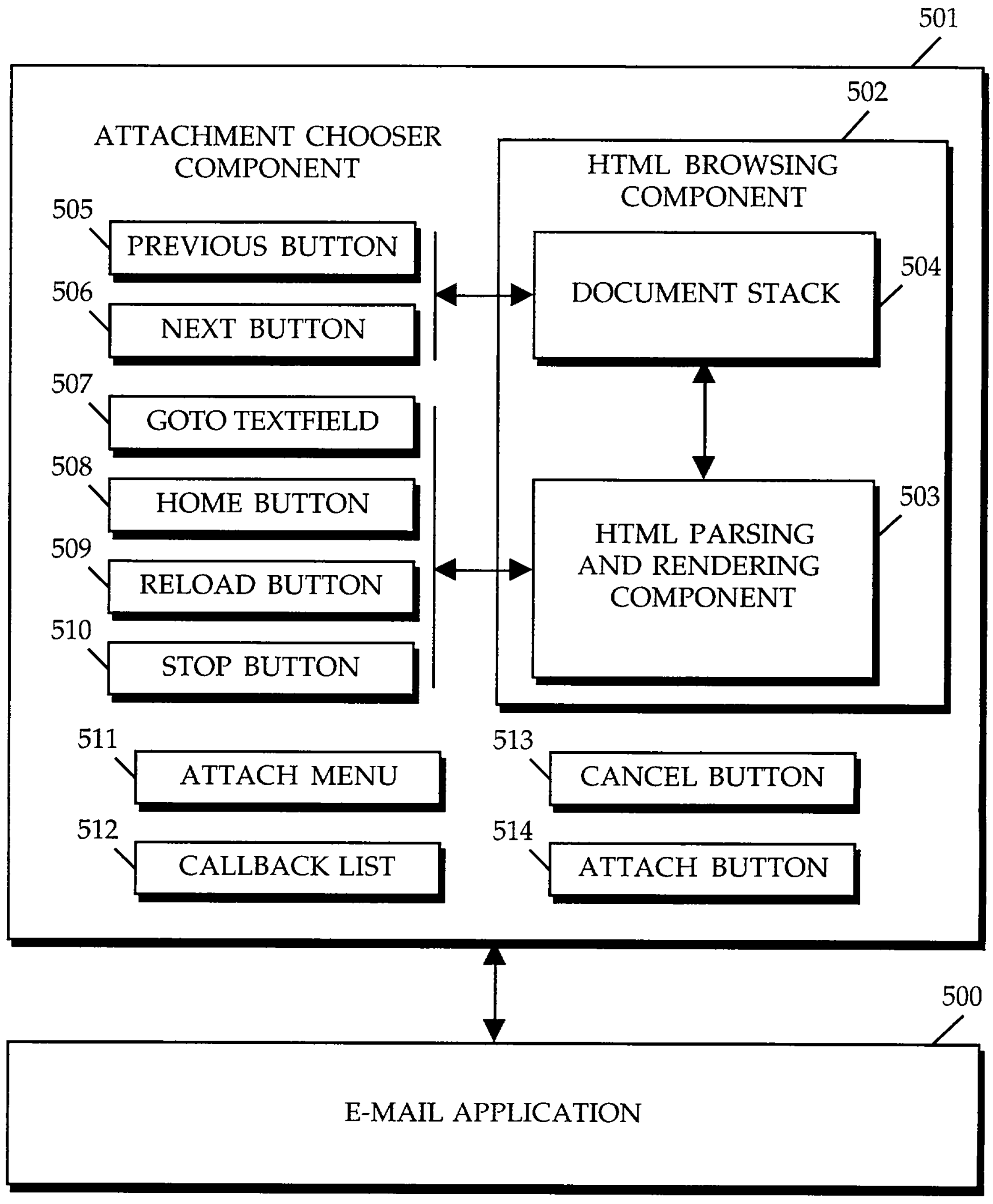
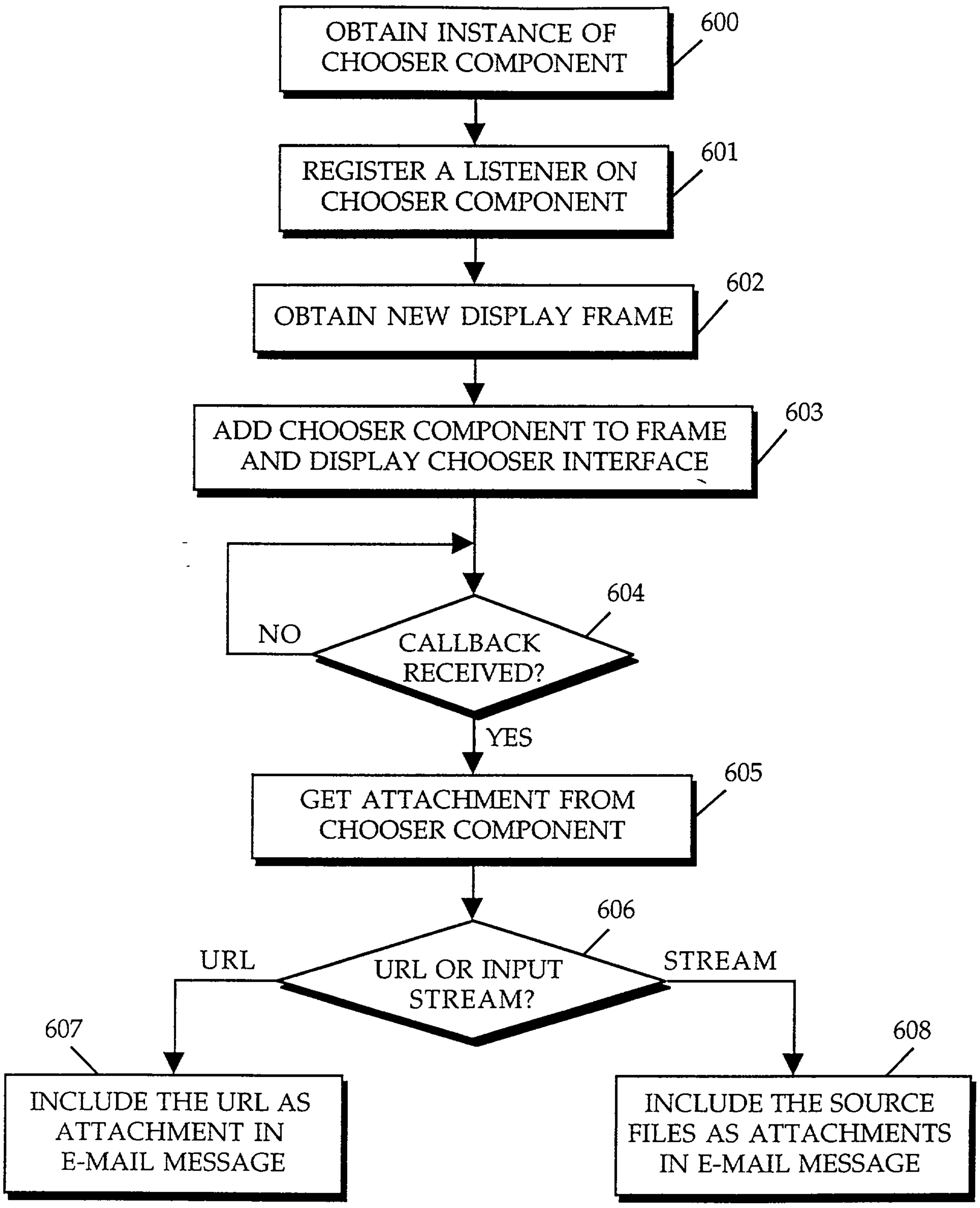


FIGURE 5B

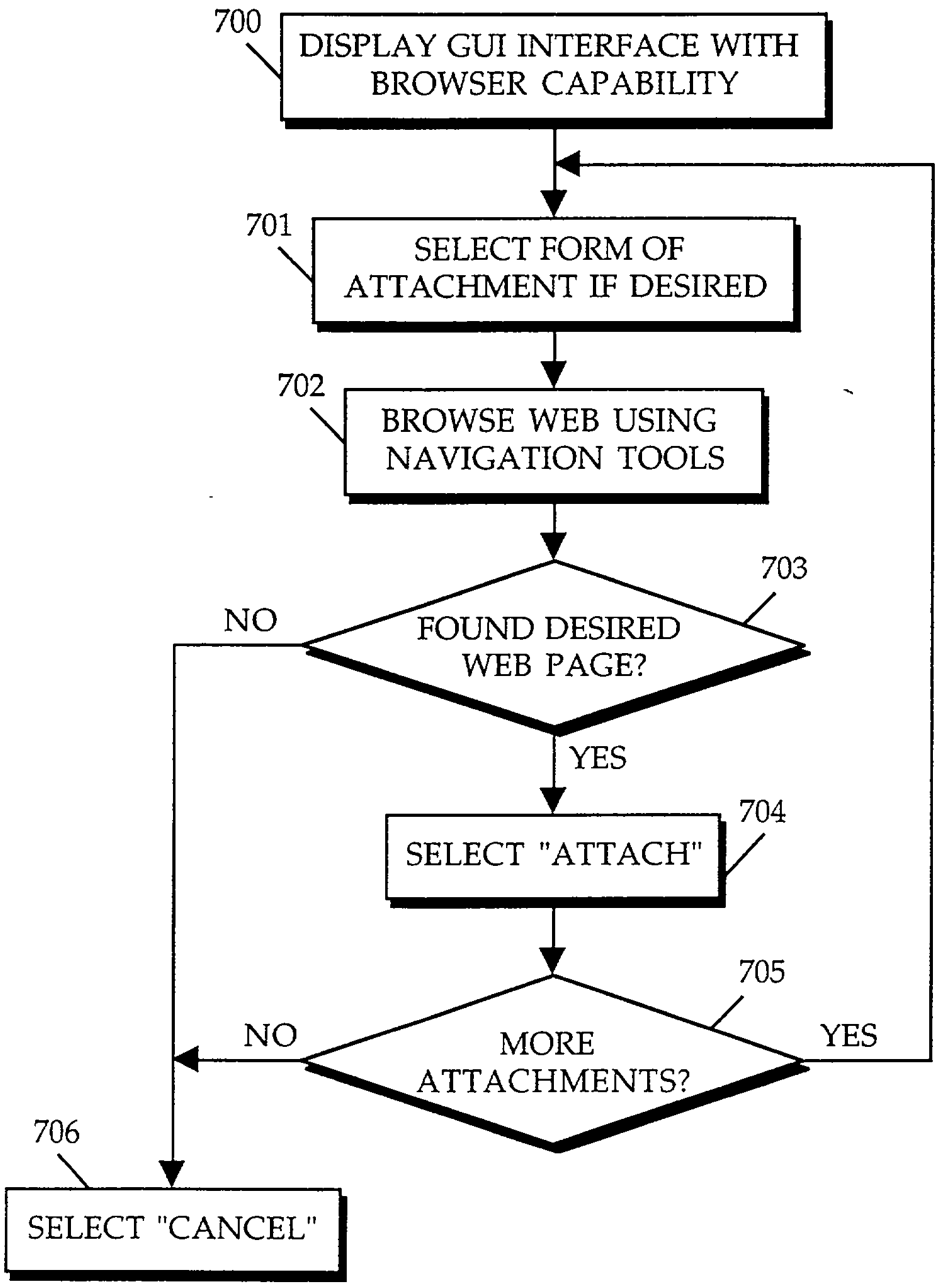
09/324095 0



OBTAIN INSTANCE OF CHOOSER COMPONENT
600
REGISTER A LISTENER ON CHOOSER COMPONENT
601
OBTAIN NEW DISPLAY FRAME
602
ADD CHOOSER COMPONENT TO FRAME AND DISPLAY CHOOSER INTERFACE
603
NO
CALLBACK RECEIVED?
604
YES
GET ATTACHMENT FROM CHOOSER COMPONENT
605
URL
URL OR INPUT STREAM?
606
STREAM
607
INCLUDE THE URL AS ATTACHMENT IN E-MAIL MESSAGE
608
INCLUDE THE SOURCE FILES AS ATTACHMENTS IN E-MAIL MESSAGE

FIGURE 6

09523035 .032300



700
DISPLAY GUI INTERFACE WITH BROWSER CAPABILITY
701
SELECT FORM OF ATTACHMENT IF DESIRED
702
BROWSE WEB USING NAVIGATION TOOLS
703
NO
FOUND DESIRED WEB PAGE?
YES
704
SELECT "ATTACH"
705
NO
MORE ATTACHMENTS?
YES
706
SELECT "CANCEL"

FIGURE 7

09/046,884 032398

CERTIFICATE OF MAILING

This is to certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as Express Mail Label No. EM 484 490 103 US **in an envelope addressed to: Assistant Commissioner for Patents, Washington, D.C. 20231 on:**

March 23, 1998

Lillian E. Rodriguez 3-23-98

Signature **Date**

Lillian E. Rodriguez

05045784.032358

Our Ref.: 83000.1007/P2975

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below, next to my name,

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled

METHOD AND APPARATUS FOR SELECTING ATTACHMENTS

the specification of which

XXX is attached hereto.
_____ was filed on ____________________ as
Application Serial No. ________________
and was amended on ________________
(if applicable)

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above. I do not know and do not believe that the same was ever known or used in the United States of America before my invention thereof, or patented or described in any printed publication in any country before my invention thereof or more than one year prior to this application, that the same was not in public use or on sale in the United States of America more than one year prior to this application, and that the invention has not been patented or made the subject of an inventor's certificate issued before the date of this application in any country foreign to the United States of America on an application filed by me or my legal representatives or assigns more than twelve months prior to this application.

I acknowledge the duty to disclose information which is material to the examination of this application in accordance with Title 37, Code of Federal Regulations, §1.56(a).

I hereby claim foreign priority benefits under Title 35, United States Code, §119, of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Prior Foreign Application(s)			Priority Claimed	
(Number)	(Country)	(Day/Month/Year Filed)	Yes	No
(Number)	(Country)	(Day/Month/Year Filed)	Yes	No
(Number)	(Country)	(Day/Month/Year Filed)	Yes	No

I hereby claim the benefit under Title 35, United States Code, §120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, §112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, §1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

(Application Serial No.)	(Filing Date)	(Status -- patented, pending, abandoned)
(Application Serial No.)	(Filing Date)	(Status -- patented, pending, abandoned)

I hereby appoint HECKER & HARRIMAN, a firm including: Gary A. Hecker, Reg. No. 31,023; J.D. Harriman II, Reg. No. 31,967; Frank M. Weyer, Reg. No. 33,050; Carole A. Quinn, Reg. No. 39,000; Ross D. Snyder, Reg. No. 37,730; and Jason S. Feldmar, Reg. No. 39,187 with offices located at 2029 Century Park East, Suite 1600 Los Angeles, California 90067, telephone (310) 286-0377, and of SUN MICROSYSTEMS, INC.: Kenneth Olsen, Reg. No. 26,493; Matthew C. Rainey, Reg. No. 32,291; Erwin J. Basinski, Reg. No. 34,773; Timothy J. Crean, Reg. No. 37,116; Philip J. McKay, Reg. No. 38,966; Robert S. Hauser, Reg. No. 37,847; and Patrick J.S. Inouye, Reg. No. 40,297 with offices located at 901 San Antonio Road, Palo Alto, California 94303, as my attorneys with full power of substitution and revocation, to prosecute this application and to transact all business in the Patent and Trademark Office connected herewith.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under §1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full Name of Sole Inventor Kapono D. Carter

Inventor's Signature [illegible] Date March 23, 1998

Residence San Jose, California (City, State) Citizenship U.S.A. (Country)

Post Office Address 5255 Firenze Ct.
San Jose, California 95138

09047845.032398

CERTIFICATE OF MAILING

This is to certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as Express Mail Label No. EM 484 490 103 US ***in an envelope addressed to: Assistant Commissioner for Patents, Washington, D.C. 20231 on:***

March 23, 1998

[illegible] 3-23-98
Signature ***Date***
Lillian E. Rodriguez

83000.1007P2975